10-16-00

A

JC948 U.S. PTO
10/13/00

Please type a plus sign (+) inside this box → [+]

PTO/SB/05 (4/98)
Approved for use through 09/30/2000. OMB 0651-0032
Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

UTILITY PATENT APPLICATION TRANSMITTAL

(Only for new nonprovisional applications under 37 C.F.R. § 1.53(b))

Attorney Docket No.	0102396-00010
First Inventor or Application Identifier	Stakutis
Title	LOW OVERHEAD METHODS AND APPARATUS
Express Mail Label No.	EL684297128US

JC868 U.S. PTO 09/687668 10/13/00

APPLICATION ELEMENTS
See MPEP chapter 600 concerning utility patent application contents.

ADDRESS TO: Assistant Commissioner for Patents, Box Patent Application, Washington, DC 20231

1. [X] Patent Application Transmittal Form
2. [X] * Fee Transmittal Form (e.g., PTO/SB/17) (Submit an original and a duplicate for fee processing)
3. [X] Specification [Total Pages 59]
   - Description (No. of Sheets: 20)
   - Claims (No. of Sheets: 7)
   - Abstract (No. of Sheets: 1)
   - Appendix (No. of Sheets: 30) A+B, corr 10/13/00
   - Other: cover page (No. of Sheets: 1)
4. [X] Drawing(s) (35 U.S.C. 113) [Total Sheets 4]
5. Oath or Declaration [Total Pages 5]
   - a. [ ] Newly executed (original or copy)
   - b. [X] Unexecuted
   - c. [ ] Copy from a prior application (37 C.F.R. § 1.63(d)) (for continuation/divisional with Box 16 completed)
     - i. [ ] DELETION OF INVENTOR(S) Signed statement attached deleting inventor(s) named in the prior application, see 37 C.F.R. §§ 1.63(d)(2) and 1.33(b).

*NOTE FOR ITEMS 1 & 13: IN ORDER TO BE ENTITLED TO PAY SMALL ENTITY FEES, A SMALL ENTITY STATEMENT IS REQUIRED (37 C.F.R. § 1.27), EXCEPT IF ONE FILED IN A PRIOR APPLICATION IS RELIED UPON (37 C.F.R. § 1.28).

6. [ ] Microfiche Computer Program (Appendix)
7. Nucleotide and/or Amino Acid Sequence Submission (if applicable, all necessary)
   - a. [ ] Computer Readable Copy
   - b. [ ] Paper Copy (identical to computer copy)
   - c. [ ] Statement verifying identity of above copies

ACCOMPANYING APPLICATION PARTS

8. [ ] Assignment Papers (cover sheet & document(s))
9. [ ] 37 C.F.R. §3.73(b) Statement (when there is an assignee) [ ] Power of Attorney
10. [ ] English Translation Document (if applicable)
11. [ ] Information Disclosure Statement (IDS)/PTO-1449 [ ] Copies of IDS Citations
12. [ ] Preliminary Amendment
13. [X] Return Receipt Postcard (MPEP 503) in duplicate (Should be specifically itemized)
14. [ ] * Small Entity Statement(s) (PTO/SB/09-12) [ ] Statement filed in prior application, Status still proper and desired
15. [ ] Certified Copy of Priority Document(s) (if foreign priority is claimed)
16. [ ] Other: ....

17. If a CONTINUING APPLICATION, check appropriate box, and supply the requisite information below and in a preliminary amendment:
[ ] Continuation [ ] Divisional [X] Continuation-in-part (CIP) of prior application No: 09/309,453
Prior application information: Examiner T. Pardo Group / Art Unit: 2771
For CONTINUATION or DIVISIONAL APPS only: The entire disclosure of the prior application, from which an oath or declaration is supplied under Box 4b, is considered a part of the disclosure of the accompanying continuation or divisional application and is hereby incorporated by reference. The incorporation can only be relied upon when a portion has been inadvertently omitted from the submitted application parts.

18. CORRESPONDENCE ADDRESS

[ ] Customer Number or Bar Code Label 21125 (Insert Customer No. or Attach bar code label here) or [X] Correspondence address below

Name	David J. Powsner		
Address	Nutter, McClennen & Fish, LLP		
	One International Place		
City	Boston	State	MA
Zip Code	02110-2699		
Country	US	Telephone	617-439-2717
Fax	617-310-9717		

Name (Print/Type)	Michael I. Falkoff	Registration No. (Attorney/Agent)	30,833
Signature	[illegible]	Date	10/13/00

Burden Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Box Patent Application, Washington, DC 20231.

EL684297128US

PTO/SB/17 (12/99)
Approved for use through 09/30/2000. OMB 0651-0032
Patent and Trademark Office: U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

FEE TRANSMITTAL for FY 2000

Patent fees are subject to annual revision.
Small Entity payments must be supported by a small entity statement, otherwise large entity fees must be paid. See Forms PTO/SB/09-12. See 37 C.F.R. §§ 1.27 and 1 28.

TOTAL AMOUNT OF PAYMENT ($) 898.00

Complete if Known	
Application Number	
Filing Date	10/13/00
First Named Inventor	Stakutis
Examiner Name	T. Pardo
Group / Art Unit	2771
Attorney Docket No.	0102396-00010

METHOD OF PAYMENT (check one)

1. ☐ The Commissioner is hereby authorized to charge indicated fees and credit any overpayments to:

Deposit Account Number: 141449

Deposit Account Name: Nutter, McClennen & Fish LLP

☐ Charge Any Additional Fee Required Under 37 CFR §§ 1 16 and 1.17

2. ☐ **Payment Enclosed:** ☐ Check ☐ Money Order ☐ Other

FEE CALCULATION

1. BASIC FILING FEE

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description	Fee Paid
101	690	201	345	Utility filing fee	$710.00
106	310	206	155	Design filing fee	
107	480	207	240	Plant filing fee	
108	690	208	345	Reissue filing fee	
114	150	214	75	Provisional filing fee	

SUBTOTAL (1) ($) 710.00

2. EXTRA CLAIM FEES

			Extra Claims		Fee from below		Fee Paid
Total Claims	26	- 20** =	6	X	18	=	108
Independent Claims	4	- 3** =	1	X	80	=	80
Multiple Dependent					0	=	0

***or number previously paid, if greater; For Reissues, see below*

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description
103	18	203	9	Claims in excess of 20
102	78	202	39	Independent claims in excess of 3
104	260	204	130	Multiple dependent claim, if not paid
109	78	209	39	** Reissue independent claims over original patent
110	18	210	9	** Reissue claims in excess of 20 and over original patent

SUBTOTAL (2) ($) 188.00

FEE CALCULATION (continued)

3. ADDITIONAL FEES

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description	Fee Paid
105	130	205	65	Surcharge - late filing fee or oath	
127	50	227	25	Surcharge - late provisional filing fee or cover sheet.	
139	130	139	130	Non-English specification	
147	2,520	147	2,520	For filing a request for reexamination	
112	920*	112	920*	Requesting publication of SIR prior to Examiner action	
113	1,840*	113	1,840*	Requesting publication of SIR after Examiner action	
115	110	215	55	Extension for reply within first month	
116	380	216	190	Extension for reply within second month	
117	870	217	435	Extension for reply within third month	
118	1,360	218	680	Extension for reply within fourth month	
128	1,850	228	925	Extension for reply within fifth month	
119	300	219	150	Notice of Appeal	
120	300	220	150	Filing a brief in support of an appeal	
121	260	221	130	Request for oral hearing	
138	1,510	138	1,510	Petition to institute a public use proceeding	
140	110	240	55	Petition to revive - unavoidable	
141	1,210	241	605	Petition to revive - unintentional	
142	1,210	242	605	Utility issue fee (or reissue)	
143	430	243	215	Design issue fee	
144	580	244	290	Plant issue fee	
122	130	122	130	Petitions to the Commissioner	
123	50	123	50	Petitions related to provisional applications	
126	240	126	240	Submission of Information Disclosure Stmt	
581	40	581	40	Recording each patent assignment per property (times number of properties)	
146	690	246	345	Filing a submission after final rejection (37 CFR § 1.129(a))	
149	690	249	345	For each additional invention to be examined (37 CFR § 1.129(b))	

Other fee (specify) ____________________

Other fee (specify) ____________________

Reduced by Basic Filing Fee Paid **SUBTOTAL (3)** $0.00

SUBMITTED BY				Complete *(if applicable)*	
Name *(Print/Type)*	Michael L. Falkoff	Registration No. *(Attorney/Agent)*	30,833	Telephone	617-439-2879
Signature				Date	

WARNING:

Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

Burden Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time you are required to complete this form should be sent to the Chief Information Officer, Patent and Trademark Office, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Assistant Commissioner for Patents, Washington, DC 20231.

EL684297128US

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Patent Application for

LOW OVERHEAD METHODS AND APPARATUS FOR SHARED ACCESS STORAGE DEVICES

Inventors

Christopher J. Stakutis
a citizen of the United States
residing at 85 Partridge Lane
Concord, MA 01742

Kevin M. Stearns
a citizen of the United States
residing at 272 Albion St., Apt. 17
Wakefield, MA 01880

Background of the Invention

This application is a continuation-in-part of United States Patent Application Serial No. 09/309,453, filed on May 11, 1999, which is a continuation of United States Patent Application Serial No. 09/002,266, filed on December 31, 1997, which issued on September 7, 1999, as United States Patent No. 5,950,203, the teachings of all of which are incorporated herein by reference.

The invention pertains to digital data processing and, more particularly, to the sharing of disk drives and other storage devices on a networked digital data processing system. The invention has application, for example, in the processing of video, graphics, database and other files by multiple users or processes on a networked computer system.

In early computer systems, long-term data storage was typically provided by dedicated storage devices, such as tape and disk drives, connected to a central computer. Requests to read and write data generated by applications programs were processed by special-purpose input/output routines resident in the computer operating system. With the advent of "time sharing" and other early multiprocessing techniques, multiple users could simultaneously store and access data -- albeit only through the central storage devices.

With the rise of the personal computer and PC-based workstations in the 1980's, demand by business users led to development of interconnection mechanisms that permitted otherwise independent computers to access one another's storage devices. Though computer "networks" had been known prior to this, they typically permitted only communications, not storage sharing.

Increased power of personal computers and workstations is now opening ever more avenues for their use. Video editing applications, for example, have until recently demanded specialized video production systems. Now, however, such applications can be run on high-end personal computers. By coupling these into a network, multiple users can share and edit a single video work. Reservation systems and a host of other applications also commonly provide for simultaneous access to large files by multiple parties or processes. Still other tasks may require myriad small files to be accessed by multiple different parties or processes in relatively short or overlapping time frames.

Network infrastructures have not fully kept pace with the computers that they interconnect. Though small data files can be transferred and shared quite effectively over conventional network interconnects, such as Ethernet, these do not lend themselves, for example, to sharing of large files. Thus, although users are accustomed to seemingly instantaneous file access over a network, it can take over an hour to transfer a sixty second video file that is 1.2 GBytes in length.

Some interconnects permit high-speed transfers to storage devices. The so-called fiber channel, for example, affords transfers at rates of up to 100 MBytes/sec -- more than two orders of magnitude faster than conventional network interconnects. Although a single storage device may support multiple fiber channel interfaces, the industry has only recently set to developing systems to permit those workstations to share such files on a storage device. Moreover, when a file is to be accessed by multiple users, the overhead of server intervention can result in loss of speed advantages and efficiencies otherwise gained from the high-speed interface. In this regard, techniques such as locking, maintaining ghost files, monitoring file changes and undertaking multi-step access, check-in or housekeeping operations may be unworkable when multi-user access to many small files must be provided quickly.

In many situations, and for many specific types of networks, the coherence and security of a shared access storage system are desirable, but the nature of some of their usual storage transactions is ill-suited to such a file management protocol. For example, a web server application may commonly require hundreds or thousands of file-OPENs-per-second (FOPS) to be carried out on a small number of large graphic or web page files. Certain commercial transaction processing and reporting applications may require simultaneous access to read hundreds or thousands of files that are quite small, and some of these tasks may be carried out with tape systems, thus introducing long delays between opening of the files and completion of a processing task. The number of file-specific network communications involved in requesting, preauthorizing or monitoring and correcting data in these file transactions might bring a system to a standstill.

In view of the foregoing, an object of the invention is to provide improved digital data processing systems and, particularly, improved methods and apparatus of high-speed access to, and sharing of, disk drives and other storage devices on a networked computer system.

A related aspect of the invention is to provide such systems that achieve fast operation with files of diverse sizes.

A related aspect of the invention is to provide such systems as can be implemented with minimum cost and maximum reliability.

Yet another object of the invention is to provide such systems as can be readily adapted to pre-existing data processing and data storage systems.

Yet still another object of the invention is to provide such systems as can be readily integrated with conventional operating system software and, particularly, conventional file systems and other input/output subsystems.

Summary of the Invention

One or more of the foregoing and other desirable objects are attained by the invention, which provides low overhead methods and apparatus for accessing shared storage on a networked digital data processing system.

A system according to one aspect of the invention includes a plurality of digital data processing nodes and a storage device, e.g., a disk drive, a "jukebox," other mass storage device or other mapped device (collectively referred to herein after as "disk drive," "storage device" or "peripheral device"). First and second ones of the nodes, which may be a client and a server node, respectively, are coupled for communication over a LAN, network or other communications pathway. Both the first and the second nodes are in communication with the storage device. This can be over the same or different respective logical or physical communications pathways.

By way of non-limiting example, the first node and the second node can be a client and a server, respectively, networked by Ethernet or other communications media, e.g., in a wide area network, local area network, the Internet interconnect, or other network arrangement. The server and/or client can be connected to the storage device via a SCSI channel, other conventional peripheral device channel, such as a fibre channel, "firewire" (i.e., IEEE 1394 bus), serial storage architecture (SSA) bus, high-speed Ethernet bus, high performance parallel interface (HPPI) bus or other high-speed peripheral device bus.

The first and second nodes function in the role of client and server, respectively, with respect to aspects of file access on the storage device. Thus, for example, the second or server node can obtain from the from the storage device or otherwise maintain administrative data, such as directory trees or file maps, pertaining to the storage of files on storage device. It can do so, for example, in its role as a complete file management system for the device or by interfacing a native file system on the storage device.

The first or client node maintains a local (or cache) copy of at least a portion of the aforementioned administrative data. This can be, for example, a file map or directory sub-tree for a file accessed by the node on the storage device. The data can be obtained in the first instance by the client node using conventional protocols for access to the storage device via the server or otherwise. Functionality (referred to below, as a "file application") executing on the client node determines whether the cached data remains current or otherwise valid. If so, the client node uses that data directly, thereby, minimizing further communications with and intervention by the server node. If not, the client node obtains updated data from the server node, e.g., again using the conventional protocols.

Further aspects of the invention provide a system as described above in which the client node caches file data (i.e., whole or partial contents of a file) in lieu of, or in addition to, administrative data relating to that file. Thus, for example, in addition to caching a directory sub-tree and block map for a file of interest, the client node can cache data read from the file.

Related aspects of the invention provide a system as described above in which the server node generates notifications (e.g., "change lists") identifying changes to administrative or file data. It is on the basis of these notifications, which may be communicated to the client node node by publication or otherwise, that the client determines whether the cached data remains current or otherwise valid. By way of example, the server node can notify the client node of any changes to the storage device impacting a file opened for READ or WRITE access by the client. Likewise, where the client effects changes to a file on the storage device, it can itself publish a change list alerting the server or other nodes, so that respective portions of their local caches can be marked as invalid.

According to some aspects of the invention, a client node is automatically "registered" by the server to receive change lists for all files opened by it on the storage device. According to other aspects, the client node can register for notification of changes to selected files. Such registration, whether specified automatically or otherwise, can include a "saturation" level. In the event the specified file is changed more frequently than reflected by that level, the server node so notifies the client node which, thereafter, assumes that the relevant portions of its cache are always invalid. This obviates the need to further publish change lists for files that are "constantly" changing (i.e., changing more frequently than indicated by the saturation level).

Related aspects of the invention provide a system as described above in which the server node (or another node that publishes change lists) does not continue to publish notifications to a client node, until that client node has acknowledged that its cache is up-to-date in relevant regards. This obviates repeated publication of change lists to a client node that, for one reason or another, has not apparently responded to prior lists.

Systems of the type described above operate on the premise that large parts of the storage device's file system remain unchanged for a substantial time. Rather than servicing each request for file access by the client node, the server node publishes lists of file changes. Likewise, whenever possible, the client node utilizes its cache of administrative or other data, o answer its file needs, thereby obviating or reducing the need for interventions by or involvement of the server node. By having the client node, for example, directly mount the file system, access files and cache those files and related administrative data, the FOPS rate is improved by orders of magnitude. Network performance is not appreciably slowed in the few instances when files are found to be on the change/restricted list, since the relatively unchanging nature of the file system assures that this will generally occur less frequently.

In general, the systems of the present invention may operate with a network file management system capable of organizing and maintaining file storage and access operations in a multi-client network system. This may be a shared storage access network file management system, or LAN networking system.

The client and server nodes may include a file system of the type as described in the above-referenced United States Patent, e.g., that executes on the first and second nodes; that is capable of responding to access requests by the client node for transferring data between that node and the storage device via the server node and a first communications pathway; and that responds to selected access requests by the client node by transferring data between that node and the storage device over a possibly separate communications pathway.

These and other aspects of the invention are evident in the drawings and in the description that follows.

Brief Description of the Drawings

A more complete understanding of the invention may be attained by reference to the drawings, in which

Figure 1 depicts a storage area network of the type with which the invention may be practiced;

Figure 2 depicts a software architecture of exemplary nodes in a system according to Figure 1;

Figure 3 depicts caching and notification in a system according to the invention; and

Figure 4 depicts a client node file application operating in a system according to the invention.

Detailed Description of the Illustrated Embodiment

Figure 1 depicts a scaleable networked digital data processing system of the type used to practice the invention. The system 10 includes a plurality of nodes 12 – 24, including two server nodes 18, 20 coupled via network pathways 26, 28 to client nodes 12 – 16 and 22 – 24, as shown. Server nodes 18, 20 are additionally coupled to one another via network pathway 27.

In the illustrated embodiment, nodes 12 - 24 represent digital data processing apparatus or other devices capable of being coupled to one another in a network and, more particularly, by way of example, in a client-server configuration. Illustrated server nodes 18, 20 represent mainframe computers, workstations, personal computers, or other digital data processing apparatus capable of providing server functions in such networks and, particularly, of controlling access to shared peripheral devices, such as storage device 36. Nodes 12 - 16 and 22 - 24 likewise represent workstations, personal computers, dedicated devices, or other digital data processing apparatus that generate requests for access to such shared peripheral devices.

The network pathways 26 - 28 represent wire cable interconnects, wireless interconnects, point-to-point interconnects, Internet interconnects or other digital communications interconnects of the type known in the art. Those pathways can be configured in any configuration that permits a node 12 - 16, 20 - 24 requesting access to a shared peripheral device 36 to communicate that request to a node 18 controlling access thereto. For purposes hereof and unless otherwise evident from context, such a requesting node is referred to as a "client," regardless of its role (i.e., as a client or server) in the conventional network defined by nodes 12 - 18 and pathway 26, or nodes 20 - 24 and pathway 28. Thus, for example, node 18 could be a "client" to node 16 for purposes of sharing peripheral device 34, presuming an auxiliary connection (e.g., fiber channel) were provided between node 18 and that peripheral device.

In the illustrated embodiment, nodes 12 – 24 operate under the Microsoft Windows NT operating system, though those skilled in the art will appreciate that the nodes 12 – 24 may utilize other client and server operating systems, as well. Moreover, it will be appreciated that nodes need not utilize the same operating systems. Thus, for example, server 18 may operate as a Windows NT-based server, while server 20 operates as a UNIX-based server. The invention is therefore seen to have the advantage of permitting multiple nodes of different pedigrees, or operating system types, to access files on a common peripheral device.

With further reference to Figure 1, the nodes 12 - 24 are coupled to respective dedicated storage devices 30 - 42, as shown. Such couplings are provided by SCSI channels or other device interconnects suitable for permitting the nodes to transfer information with such devices. In addition to being coupled to their own dedicated storage devices 34, 38, nodes 16, 20 are coupled to the storage device 36 that is controlled by node 18. In the parlance of the invention, nodes 16, 20 are referred to as "clients" and node 18 is referred to as a "server."

Coupling between the clients 16, 20 and the shared peripheral device 36 can be provided by any conventional peripheral device interconnect, though, preferably, it is provided by high-speed interconnects such as fibre channel, "firewire" (i.e., IEEE 1394 bus), serial storage architecture (SSA) bus, high-speed Ethernet bus, high performance parallel interface (HPPI) bus or other high-speed peripheral device bus.

Figure 2 depicts further detail of a hardware and software architecture permitting low overhead access to files on a shared peripheral device 36 by nodes 16, 20 in a system according to the invention. Though the discussion that follows is directed to access among these devices, those skilled in the art will appreciate that the teachings can be applied equally to file access on any of storage devices 30 - 32 by any of the nodes 12 – 24 to which they are directly or indirectly coupled.

Referring to the drawing, nodes 16, 18 are coupled to one another via communications pathway 26 and to peripheral device 36 via pathways 44, 46, respectively. As noted above, pathway 44 (coupling device 18 to peripheral 36) can be a SCSI channel or other conventional peripheral device interconnects. Likewise, as noted above, pathway 46 (coupling device 16 to peripheral 36) can be a conventional peripheral device interconnect, though, preferably, is a high-speed interconnect such as fibre channel, "firewire" (i.e., IEEE 1394 bus), serial storage architecture (SSA) bus, high-speed Ethernet bus, high performance parallel interface (HPPI) bus or other high-speed peripheral device bus.

Executing on node 16 are one or more applications programs 48 (e.g., including video editing programs, image analysis programs, and so forth) that generate requests for access to local and networked peripheral devices, including shared device 36. Those applications programs execute in the conventional manner under the control of an operating system, e.g., Windows NT, which includes a file system that services those access requests.

In the illustration, that file system is represented by elements 50 - 54, including "upper" file system 50, representing the Windows NT I/O Subsystem Manager and other components responsible for interfacing with applications programs 48 and for routing peripheral device access requests to the file system; "lower" file system 52, representing the Windows NT File system drivers and intermediate drivers and other components responsible for local, disk-based file systems, SCSI drivers and the like providing generic functionality to a common set of devices; and drivers 54, representing software (and hardware) components for transferring information to and from attached peripheral devices 34, 36.

Because node 16 is a client vis-à-vis the Windows NT network, the drivers also include a network redirector, such as the Windows NT LANManRedirector, that transfers access requests to and from the shared peripheral device 36 via server node 18 and pathways 26 and 44. The node 18, which includes network server component 56, handles such requests in the conventional manner of a server of a networked digital data processing system. As illustrated, node 18 also includes a file system, comprising elements 58 - 64, whose operations parallel those of components 50 - 54 on the node 16.

Though the illustrated architecture for node 16 is in accord with that dictated by Windows NT, those skilled in the art will appreciate that the invention may be embodied in devices running under other operating systems, as well.

The illustrated system allows multiple client nodes to access files on a shared peripheral device with minimal overhead and with coordination of a server node for that device. A server node is used in this approach to notify the clients of changes to administrative (or meta) data and file data cached by them. A direct connection, or "directly attached disk connect" can be provided between each node and the shared device to permit certain operations, e.g., bulk reads and writes, to be accomplished directly between the nodes and the peripheral device. The illustrated embodiments provide this capability through communications pathways such as pathway 46 and through filter drivers 66, 68 incorporated into the file systems. The direct connection may be physically separate from the network connection provided between the nodes or it may be logically separate, e.g., carried by the same physical conductor set as the network connection via high-speed switches and/or pathways.

In the discussion that follows, unless otherwise evident from context, the term "file system" refers in this context to the combined operation of the nodes' native file systems (e.g., comprising elements 50 - 54 and 56 - 64) and of the network server, e.g., 56., and file system, e.g., 56 - 64, of the node to which the shared peripheral device is assigned and of the file system, e.g., 50 - 54 of the node that shares that device. The storage device 110 stores data, e.g., files, records, data structures, or other ordered data, any of which shall generically be referred to herein as a “file”.

The context and operation of the illustrated system will be better understood following a brief discussion of an implementation of shared storage access in a network. In a conventional network shared-access system, the server and the meta data controller (MDC) for a storage device is the only unit that hard-mounts a volume, lays down a file system and manages that file system for that storage device. In one respect, a server MDC is somewhat like a file-server for a volume; other machines may use conventional networking to access the volume and thus are authenticated and controlled in the traditional networking model. However, server communications on the network involve only meta data (file open, security, allocation information, etc). The meta data is transferred via the server over the network the actual file payload or data content may be transferred directly between the client nodes and the storage elements.

This arrangement, while greatly speeding up the shared processing of large files, has a certain amount of overhead associated with it. Generally each node or client must effect certain communications through the MDC, requesting meta data and reporting file closing and change data. The server may be the only node that can write files, employing a single cache for storing and monitoring all outstanding maps and files, and requiring that the data be passed through the server.

When files are large, e.g., over 100 kilobytes, the overhead of the multiple required network communications for authorization, tracking and housekeeping may appear negligible. However, for file sizes under 10 kilobytes, a UNIX server may be expected to perform 15,000 file opens per second (FOPS), and a conventional network-based file system may approach 1500 FOPS. Interposition of a shared access file management system, or a distributed lock system, may reduce the rate even below this normal network speed, e.g., to only several hundred FOPS.

This slowing down may appear unavoidable, because when many smaller files are opened in a short time, any cached data and meta data for these files may quickly become inaccurate, requiring extensive housekeeping and/or communications over the network. Thus, the ability to provide shared access, while offering dramatic efficiency for large files, carries a penalty that becomes more onerous for accessing smaller files. The disparity between FOPS rates achievable with and without a shared storage file management system is especially great when one considers tasks such as simple open-to-read file accesses, or other circumstances commonly arising in web server applications, information mining and networked processing situations.

This problem is addressed in the illustrated embodiment by providing caches for meta data and data in the client nodes and by publishing messages that allow the caches to be maintained and used as long as possible for direct file access. It may in further aspects minimize messaging required to maintain those caches. Briefly, a client node OPENing or otherwise requesting a file receives meta data, i.e. block address data, from the server or file system meta data controller. The client stores this in a local cache or in a cache otherwise dedicated to that node. The client may use the cached data directly, e.g., to access the desired file(s) on the storage device. File data obtained as a result of those accesses is cached as well, by the clients. The server, which operates as, or interfaces with, a file system meta data controller to perform housekeeping necessary for file coherence, storage allocation efficiency and the like, publishes to the client nodes change lists indicating whether changes have been made to the file system that may affect meta data or file data cached by the client nodes. A file application on each client consults the change list to determine whether the client may utilize portions of its cache or fresh meta data or file data must be obtained (e.g., from the server or via direct access to the storage device.)

Figure 3 schematically illustrates a system 100 in accordance with the present invention operating to increase speed of a shared access network system. As shown, a system of the invention includes a storage unit 110, and a host 120 connected to the storage unit and operating with a file storage protocol to permit and control access to at least some of the data in the storage unit 110. The system further contains client nodes 130, e.g., user workstations or terminals in a network, of which one client node is illustrated. Those skilled in the art will readily appreciate that server 120 corresponds, for example, to server 18 of Figure 1; storage device 110, to peripheral device 36; network 100 to networks 26 – 28; and nodes 130 to client nodes 12 – 24.

Initially, a client node 130 requests access to a file, e.g., by a file OPEN operation, and the server node 120 provides meta data. This may include, for example, an identification of one or more particular disks, offsets, sector or block information and the like that allows the node 130 to directly mount the storage device 110 and access the file contents in the collection of block storage locations 115 for that file. The node 130 stores this meta data to local cache 130a and uses it to access the file, filling out the cache 130a with further meta data and file data acquired in the process. Figure 3 schematically shows a stored group of blocks of data 115 in the storage unit 110 corresponding to a requested file , and the corresponding locally cached data 116' which reside in the client 130 cache memory once the file has been opened. The locally cached meta data in cache 130a, e.g., block addresses, offsets, network path, directory or other "map" for the file in storage, is indicated schematically by a shaded portion. The cached data may also include (if the node 130 has recently opened the file) all or portions of the contents of the file (referred to above as "file data".) The local cache may be constructed in a conventional way as files are accessed and used in the node 130, and the meta data and file data are located near or adjacent in the cache of the client node 130.

The server MDC node 120 has a local cache, which may for example include a copy of the file contents, the block addresses, and other meta data. However, the server meta data may be more extensive, including much meta data related to its MDC file management and message coordination functions. Furthermore, the server file copy (if any) may be resident only while the server is coordinating transactions related to the file; generally, when an accessing node has closed the requested file, and other transactions intervene from additional client nodes, server cache will be taken over by current processing and monitoring tasks.

The nodes 130 may connect over a network 125 such as a LAN to a file management system 126 associated with the storage device 110. The file management system in a shared storage access network system may involve a native meta data controller (MDC) 126 together with an interface layer on node 120 that allows the server 120 to aquire map data from the storage device and coordinate the operations that occur over the network 125 with the file and storage unit management operations that must be carried out in the storage device 110.

In operation of the present invention, the client node caches 130a may independently subsist, and a file access application running on the client 130 may access data and meta data, as appropriate when the node 130 again requires access to the file. Thus, as shown in Figure 4, the file application may answer a file open request within the node 130 by simply providing the file contents 116 still present in the locally cached data 116', or by utilizing the corresponding meta data cached in 116' to directly access the storage device 110.

The server MDC 120 may effect this ongoing access only indirectly, which it does by the mechanism of publishing a change list that is transmitted to the client node 130 and that identifies the files or meta data that have changed and may be no longer valid. The client copy of the change list 131 is stored on the node 130, and a file application 140 on the client node 130 in turn consults the change before determining whether to use the cached data, or meta data, or to revert to carrying out a file transaction through the server 120.

Thus, in the present system, the client machines use ordinary networking to directly mount the volumes and open files. However, as files or meta data change, portions of the locally cached directory are restricted by the change lists. The local file application may automatically revert to the shared access protocol for requesting meta data from the server 120, or it may implement a decision algorithm that determines whether the locally cached data may still be used. From another perspective, the direct-access system of the present invention employs a shared-access MDC/server file management protocol, but extends the period of use or direct access, and greatly speeds up the accession of files by allowing client nodes to each maintain a natural cache with meta data and directly mount the file system until or unless the cache becomes invalid or suspect. Figure 4 illustrates such an embodiment of the invention, operating with a SAN MDC server node to also permit host-mediated accesses, but to generally minimize the instances when this will be necessary. It will be understood that the file application running on the client may intercept reads and writes, and issue those directly to the disk elements, while coordinating with the host MDC (regarding caching) and asking the MDC for the list of physical blocks (i.e., a map) for each desired file. In accordance with a principal aspect of the present invention, such structure, if provided, is not the default structure. Instead, the client nodes are permitted to maintain data maps and directly mount the file system except for a limited number of changed files. After an initial file access, the server MDC 120 functions primarily to mediate file access in the small number of cases where intervening file changes or storage reallocations have superceded the locally cached file data and meta data.

In further embodiments, the server MDC 120 may further effect communications with nodes 130 over the network to enable local file applications to remove locally cached data or prevent its reappearance when there have been intervening file changes, or even to reconstitute or correct local cached meta data.

This operation is indicated generally in Figure 4. Notifications from node 120 over the network 125 between the MDC 120 and the client node 130 form a change list 131 at each client 130. A file application 140 running on the client intercepts file READ or WRITE calls and performs a check 141 on each file request to determine whether the file is on the change list 131. If not, the client proceeds to access the file directly using its locally-cached content if present ,or applying the locally-cached meta data to access the desired blocks in storage 110. Otherwise, that is, if the file does appear on the change list, the file application may proceed to a further determination 142, to determine whether direct access is nonetheless appropriate. This allows a local or even a context-dependent determination to be made, for example, on whether a locally cached file copy is still the appropriate version for the task at hand (for example if the listed changed blocks are outside the region to which access is desired). In lieu of making such further determination 142, the client file application may simply be configured to proceed to obtain the file by a separate request to the node 120, without further inquiry. In that case, the client 130 may, for example, simply request the file from the node 120 through the SAN protocol, i.e., and thus acquire fresh meta data or data block maps to locate the changed file data locations in storage.

In various further embodiments, the client file system may undertake communications and local processing to either purge change-listed data and meta data from cache, or to rewrite cache to update the necessary access data.

A more detailed understanding of one suitable SAN implementation of a shared access file management system made by IBM may be had from the attached APPENDIX A, which describes functions and procedures for interfacing between a commercial file system storage unit and its file management system meta data controller (or native storage unit file system meta data controller) , denoted FSMDC and the SANergy network shared access layer. APPENDIX B attached hereto further describes the higher level communications and control functions of the SANergy system, i.e., the interface protocols and necessary data and messaging structures for effecting such shared access and interfacing with the different equipment and system layers. Such a system is replaced by (in some embodiments of the present invention), or preferably augmented by (in other embodiments), the direct mount system as set forth above herein, that operates without prior authorizations for some or all data accesses, and allows clients 130 to directly mount the file system for accessing files under many conditions, using local fully cached storage data.

As noted above, to request a file, the client 130 passes a file request to a file access application 140 running on the client that consults the list 131, and either directly accesses the storage unit 110 (if the desired file is not on list 131) or implements a further decision or the MDC-mediated access protocol (if the file is on list 131). The restriction list 131 may be, briefly, a list of files, or directories or data blocks, which are not to be accessed via locally cached data. It may include also restrictions due to user authority limitations or the like that are specific to a particular client node, or to a class of client nodes. Preferably, however, the notification is simply a change list, published and updated by the MDC 120, that indicates specific portions of the local cache that may be superceded or inaccurate due to changes in the stored file. The notifications and list may be specific to the client 130 (when, for example a specific node is only configured for working with data from specific directories or has limited authority to access data), or it may be identical for several or for all client nodes.

The "server" MDC 120 may monitor the file storage to detect all file writes or identify changes in files and their storage addresses, to publish the change lists 131, which are received and cached by the client nodes 130. Such monitoring and determination of change lists, may for example be effected in a relatively straightforward way for the SANergy network protocol and an underlying FSMDC interface described in APPENDICES A and B. This assures that the node 130 does not use out-dated locally-cached meta data or data, and that the MDC is able to maintain integrity and coherence of data files that have been moved or modified. However, the invention is not limited to SANergy-type file management systems, but may be advantageously applied to any network file system to integrate direct mounting of the storage device and greatly enhanced speed for a substantial portion of the file access transactions. Thus, the invention extends a conventional file management system.

Applicant has identified this file system enhancement by the acronym ZOOM, denoting zero-overhead, zero meta data, since in a simple implementation the system entirely dispenses with network transfers of an extra layer of meta data that would be used for coordination between the nodes, file versions and required transactions through the MDC for routine file accesses, and relies on locally cached meta data for directly mounting the file system, or on locally cached file data. In general, it will be very efficient for the client to resort to a server-mediated access protocol such as the SANergy system, for accessing the few needed files which may appear on the list 131.

When used in conjunction with a shared access file system as described in the aforesaid U.S. patent, the file manager may include a bypass mechanism, which executes on at least the client node, to intercede in the response to at least selected input/output, or access, requests generated by that node, and transfer data designated by such requests directly between the client node and the storage device, in lieu of transferring that data via the server. Such transfers by the bypass are made using the administrative information maintained by the file system relating to storage of such data on the peripheral device. The bypass can intercede in response to requests by the applications programs executing on the client node to read or write data on the peripheral device. Rather than permitting the file system to transfer that data via the server node and network, the bypass transfers it directly to the peripheral device. A further understanding of these and other aspects of this aspect of the system may be attained by reference to the aforesaid patent, the teachings of which (as noted above) are incorporated herein by reference.

The foregoing description sets forth methods and apparatus meeting the objects identified above. In a principal aspect the directly-mounted file system approach of the present invention inverts a conventional file management tenet by requiring notifications of changes be sent to the client nodes, rather than using centrally-monitored change data as the basis for a centrally-implemented housekeeping and control process. The change list notification procedure of the present invention is compatible with a great number of file management systems, and may increase FOPS rates by one or more orders of magnitude for the great preponderance of file accesses. Moreover, in relying on client file applications to initially determine the access protocol, the present invention allows task- or client- specific determinations as to the level of change or corruption that may occur without detriment, offering greater flexibility for diverse applications. Those skilled in the art will appreciate that the illustrated embodiment is shown and described merely by way of example and that other embodiments incorporating changes therein fall within the scope of the invention, of which we claim:

1. A digital data processing system with improved access to information stored on a storage device, said system comprising a plurality of first nodes and a second node coupled to one another over a communications pathway, the second node being coupled to the storage device for determining meta data including block address maps to file data in the storage device, and the first nodes being configured for accessing file data from the storage device using said meta data, wherein said system comprises

at least one first node that caches data including meta data for a file accessed by said first node

a file application on said first node configured to get requested file data by accessing said cached data for the file, and

a file notification system that sends a file change notification to said first node indicating changes affecting the cached data,

wherein the file application on the first node inspects the change notification to determine whether to get the requested file data directly using said cached data, whereby file accesses may be effected for an extended time with data locally cached at first nodes of the system.

2. The digital data processing system of claim 1, wherein the file application on said first node determines whether requested file data is subject to a change notification, and if so makes a further determination whether cached data at said first node remains valid for the requested file data.

3. The digital data processing system of claim 1, wherein the file application on said first node,

   i) determines whether requested file data is subject to a change notification, and

   ii) applies the cached meta data to directly mount the storage device to access the requested file when the cached data is not subject to a change notification.

4. The digital data processing system of claim 2, wherein the file application on said first node further determines whether

   i) to directly access the file data by applying cached meta data associated with the file to directly mount the storage device, or

   ii) to issue a file request to the second node for valid file access meta data or data.

5. The digital data processing system of claim 1, wherein the file notification system issues client-specific notifications limited to directories or portions of the file system that are to be accessed by each client.

6. The digital data processing system of claim 1, wherein the file notification system includes an interface layer with a storage system meta data controller for maintaining or acquiring administrative information pertaining to file size and storage locations.

7. The digital data processing system of claim 1, wherein the file notification system runs on the second node and interfaces with a file system meta data controller to detect changes in file system storage data, issuing a file change notice in response thereto.

8. The digital data processing system of claim 1, wherein the file notification system limits number of change notifications for a given file to first n changes that occur, where n is a positive integer.

9. The digital data processing system of claim 1, wherein the file application on the first node implements a decision algorithm to determine whether to apply cached data for a requested file when the requested file is subject to a change notification.

10. The digital data processing system of claim 1, wherein the file application on the first node intercepts reads and writes, and issues those directly to the storage device while exchanging messages over the communications pathway to permit coordinate file system management tasks performed by the second node.

11. The digital data processing system of claim 10, wherein the file system management tasks performed by the second node include publication of change data.

12. The digital data processing system of claim 1, wherein the file shared access coordination system runs on the second node and interfaces with or includes a file system meta data controller interceding in response to at least a first selected file access request applied thereto by a file application on a first node, and transferring data designated by that request between the first node and the peripheral device in accord with current meta data maintained by the file system pertaining to storage of that data on the storage device such that files may be directly transferred while maintaining file coherence and security.

13. A digital data processing system, comprising

a first node and a second node coupled for communication, a storage device coupled for communication with at least the first node,

a cache memory coupled to and associated with the the first node, the cache memory storing administrative data pertaining to files on the storage device,

the second node notifying the first node of changes to administrative data pertaining files for which the cache memory stores administrative data.

14. A digital data processing system according to claim 13, wherein the storage device is any of a disk drive, a "jukebox," other mass storage device or other mapped device.

15. A digital data processing system according to claim 13, wherein the administrative data stored by the cache includes any of a physical storage map and at least a portion of a directory pertaining to files on the storage device.

16. A digital data processing system according to claim 13, wherein the digital data processing system comprises a network having a file management system, and a file application on the first node applies administrative data in the cache memory pertaining to a file directly mount the storage device.

17. A digital data processing system according to claim 13, wherein the digital data processing system comprises a network having a file management system, and a file application applies a notification of a change of administrative data pertaining to a given file by passing a request for that file by the first node to the file management system.

18. A method of sharing storage access in a digital data processing system having a first node and a second node coupled for communication and a storage device coupled for communication with at least the first node, such method comprising the steps of

caching in a cache memory coupled to and associated with the first node, administrative data pertaining to files on the storage device,

communicating, to the first node, changes to administrative data pertaining to files for which the cache memory stores administrative data, and

determining, in the first node, whether to apply said cached data for accessing a file thereby reducing network communications..

19. A digital data processing method for improved access to information stored on a storage device, wherein the system includes a storage device, a plurality of first nodes and a second node communicating over a communications pathway, the second node being coupled to the storage device for determining meta data for accessing file data in the storage device, and the first nodes being configured for accessing file data from the storage device using said meta data, wherein said method is characterized by the steps of

caching meta data for a file accessed by said first node in a cache memory of said first node

providing a file application on said first node configured to get requested file data utilizing said cached data

storing file change notifications at said first node indicating changes that may affect the cached data, and determining, via said change notifications whether said file application may utilize the cached data for a requested file.

20. The method of claim 19, wherein the step of storing file change notifications is effected by receiving a change list published by the second node and storing the change list.

21. The method of claim 19, wherein the file application

i) determines whether requested file data is subject to a change notification, and

ii) applies the cached meta data to directly mount the storage device to access requested file when the cached data is not subject to a change notification.

22. The method of claim 19, wherein the file application on said first node operates

i) to directly access the file data by applying cached meta data associated with the file to directly mount the storage device when said cached data is not subject to a change notification, or

ii) to issue a file request to the second node when said cached data is subject to a change notification.

23. The method of claim 19, wherein the file notifications are client-node-specific notifications limited to directories or portions of the file system that are to be accessed by each client.

24. The method of claim 19, wherein the file notification system runs on the second node operates with a file system meta data controller to detect changes in file system storage data and issue file change notifications in response thereto.

25. The method of claim 24, wherein the second node limits number of change notifications for a given file to first *n* changes that occur, where *n* is a positive integer.

26. The method of claim 19, wherein the file application on the first node implements a decision algorithm to determine whether to apply cached data for a requested file.

Abstract of the Invention

A digital data processing system with improved network access to files in a storage system has a plurality of digital data processing nodes and a storage device. First or “client" nodes locally cache meta data such as maps to file data stored on the storage device and directly mount the file system using their cached data. A second "server" or MDC node connects to the storage device and mediates access to files, issuing file change notifications over the network for changed files or meta data. The second node preferably implements a SAN shared access storage system, and initially provides the maps for requested files, from which each of the first nodes builds their own cache in a natural way. In further embodiments, the client nodes may register for access to particular file areas or sub-directories, and may then receive change notifications only for their registered access areas or portions of the data tree; alternatively, the second node may determine scope of necessary notifications from the set of previously-accessed files. A file application on the client inspects change lists to determine whether to directly access (or use previously accessed file data) using the cached information, or whether a desired file must be accessed by requesting new maps from the second node through a network file sharing protocol. The system operates on the premise that large parts of the file system remain unchanged for a substantial time. Thus, rather than arbitrating each request for file access, the file notification system publishes and updates one or more lists of file changes. These may identify portions of the file directory cached at a client node that are either off-limits, or require specific determinations as to their continued usability. The local file application implements a two-stage access protocol that allows direct mounting of the file system by all or a subset of client nodes for both files and meta data, and extends the time during which file accesses by cached data may be effected. The notification system may run on a server node configured to implement shared storage by interfacing with and coordinating file access, write and block re-allocation operations of a native file management system to maintain file integrity, coherence or security, and may apply the change list to prevent local file applications from accessing restricted data as well as preventing client node use of changed file data or outdated data maps.

908645.1

EL684297128US

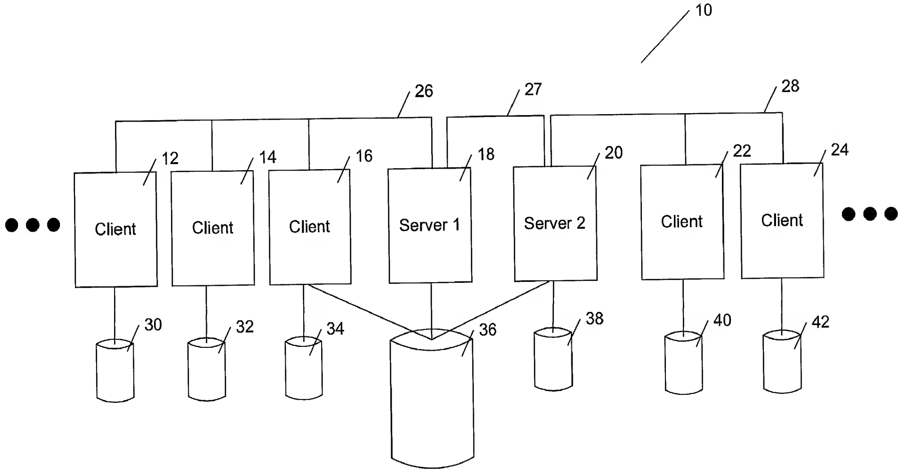


Figure 1

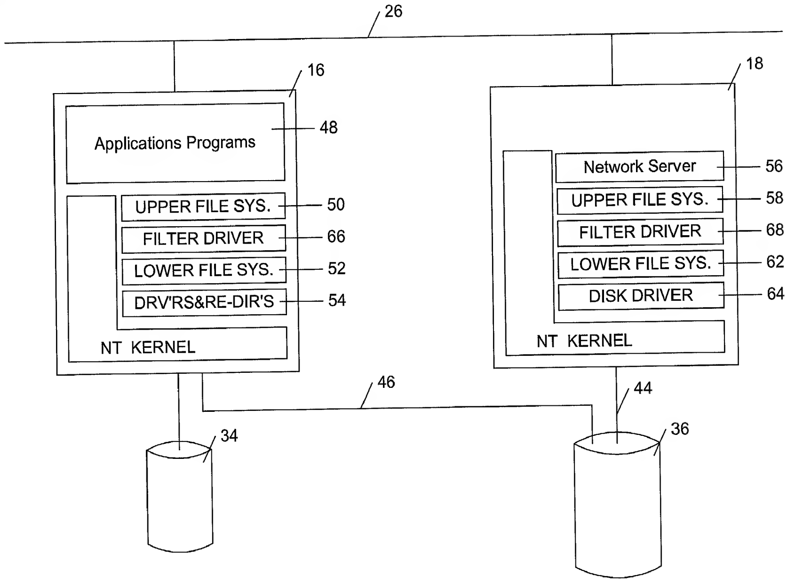

26
16
18
Applications Programs
48
UPPER FILE SYS.
50
FILTER DRIVER
66
LOWER FILE SYS.
52
DRV'RS&RE-DIR'S
54
NT KERNEL
Network Server
56
UPPER FILE SYS.
58
FILTER DRIVER
68
LOWER FILE SYS.
62
DISK DRIVER
64
NT KERNEL
46
44
34
36

Figure 2

EL684297128US

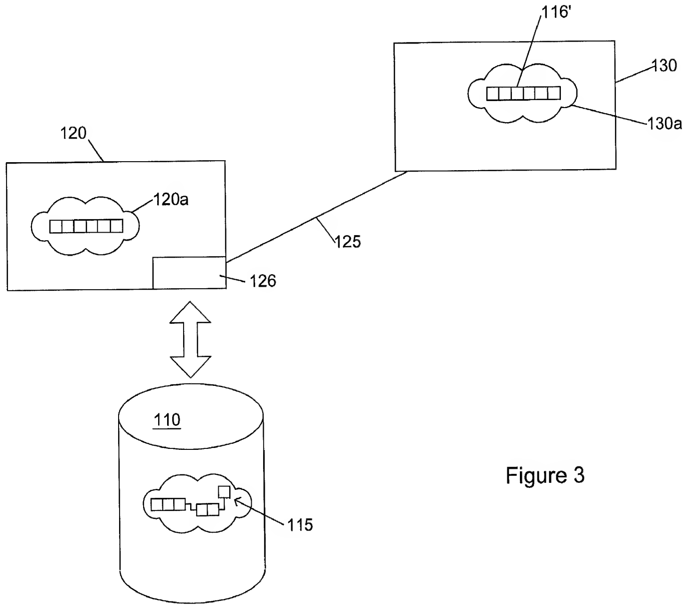


Figure 3

09687668.101300

EL684297128US

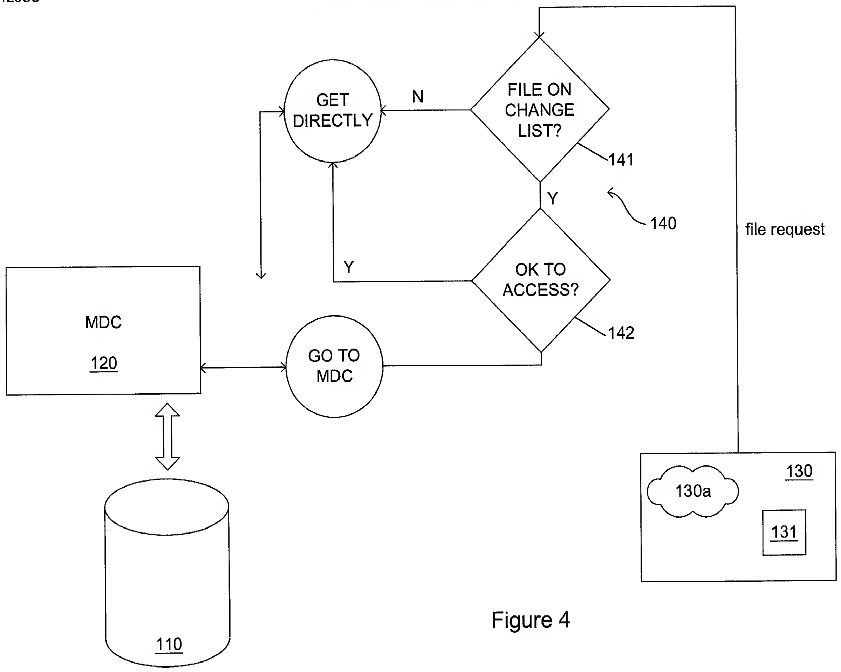


Figure 4

EL684297128US

Attorney Docket No.: **0102396-00010**

DECLARATION AND POWER OF ATTORNEY FOR UNITED STATES LETTERS PATENT APPLICATION

As a below-named inventor, I hereby declare that:

My residence, post-office address and citizenship are as stated below next to my name.

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

LOW OVERHEAD METHODS AND APPARATUS FOR SHARED ACCESS STORAGE DEVICES

the specification of which

(check one)

☒ is attached hereto.

☐ was filed on:

as Application No.:

and was amended on:
(if applicable).

In the event that the filing date and/or Application No. are not entered above at the time I execute this document, and if such information is deemed necessary, I hereby authorize and request my attorneys/agent(s) at Nutter, McClennen & Fish, LLP, One International Place, Boston, MA 02110-2699, to insert above the filing date and/or Application No. of said application.

I hereby state that I have reviewed and understand the contents of the above-identified application specification, including the claims, as amended by any amendment specifically referred to herein.

I acknowledge the duty to disclose all information known to me that is material to patentability in accordance with Title 37, Code of Federal Regulations, §1.56.

FOREIGN PRIORITY CLAIM

I hereby claim foreign priority benefits under Title 35, United States Code §119(a)-(d) of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

(check one)

☒ no such foreign applications have been filed.

☐ such foreign applications have been filed as follows:

EARLIEST FOREIGN APPLICATION(S), IF ANY FILED WITHIN 12 MONTHS (6 MONTHS FOR DESIGN) PRIOR TO THIS U.S. APPLICATION

Country	Application Number	Date of Filing (month, day, year)	Priority Claimed Under 35 USC 119
			___ Yes No ___
			___ Yes No ___
			___ Yes No ___
			___ Yes No ___
			___ Yes No ___

ALL FOREIGN APPLICATION(S), IF ANY FILED MORE THAN 12 MONTHS (6 MONTHS FOR DESIGN) PRIOR TO THIS U.S. APPLICATION

CLAIM FOR BENEFIT OF EARLIER U.S. PROVISIONAL APPLICATION(s)

I hereby claim priority benefits under Title 35, United States Code §119(e), of any United States provisional patent application(s) listed below:

(check one)

☒ no such U.S. provisional applications have been filed.

☐ such U.S. provisional applications have been filed as follows:

Application Number	Date of Filing (month, day, year)	Priority Claimed Under 35 USC 119(e)
		___ Yes No ___
		___ Yes No ___
		___ Yes No ___

CLAIM FOR BENEFIT OF EARLIER U.S./PCT APPLICATION(s)

I hereby claim the benefit under Title 35, United States Code §120, of the United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United State Code, §112, I acknowledge the duty to disclose all information that is material to patentability in accordance with Title 37, Code of Federal Regulations, §1.56, and which became available to me between the filing date of the prior application and the national or PCT international filing date of this application:

(check one)

☐ no such U.S./PCT applications have been filed.

☒ such U.S./PCT applications have been filed as follows:

Application Number	Date of Filing (month,day,year)	Status (Patented/Pending/Abandoned)
09/309,453	5/11/99	
09/002,266	12/31/97	

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment or both under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

I hereby appoint:

Ronald E. Cahill	Reg. No.	38,403	Lisa J. Michaud	Reg. No.	44,238
Maria M. Eliseeva	Reg. No.	43,328	Reza Mollaaghababa	Reg. No.	43,810
Thomas J. Engellenner	Reg. No.	28,711	David J. Powsner	Reg. No.	31,868
Michael I. Falkoff	Reg. No.	30,833	Richard J. Roos	Reg. No.	45,053
William C. Geary III	Reg. No.	31,359	Scott D. Rothenberger	Reg. No.	41,277

all of Nutter McClennen & Fish, LLP, One International Place, Boston, Massachusetts 02110-2699, jointly, and each of them severally, my attorneys at law/patent agent(s), with full power of substitution, delegation and revocation, to prosecute this application, to make alterations and amendments therein, to receive the patent, and to transact all business in the Patent and Trademark Office connected therewith.

Please mail correspondence to: David J. Powsner
at **Customer Number 021125**, whose address is:

Nutter McClennen & Fish, LLP
One International Place
Boston, Massachusetts 02110-2699

Please direct telephone calls to: David J. Powsner
at (617)439-2717.

Please direct facsimiles to: (617)310-9717

Full name of sole or first joint inventor Christopher J. Stakutis	
Inventor's Signature	Date
Residence 85 Partridge Lane, Concord, Massachusetts 01742	
Country of Citizenship US	
Post Office Address (required) same as above	

Full name of second joint inventor Kevin M. Stearns	
Inventor's Signature	Date
Residence 272 Albion Street, Apt. 17, Wakefield, Massachusetts 01880	
Country of Citizenship US	
Post Office Address (required) same as above	

914105.1

APPENDIX A

Patent Application for

LOW OVERHEAD METHODS AND APPARATUS FOR SHARED ACCESS STORAGE DEVICES

SANergy FSMDC Specification

Version 2.2.5

6 July 2000

Author & Technical Contact:
Chris Stakutis, CTO SANergy
SANergy, Westford MA
Tivoli Storage Business Unit
617 693 9770
cstakutis@tivoli.com

About This Document

This document is the Specifications for the **FSMDC** layer of the SANergy SAN-based file system sharing product.

Who Uses This Document

3rd party customers and partners
Strategy & Architecture
Technical Office(s)
Designers
Developers
Function Test
System Test
Performance
Legal Counsel
National Language Technical Center (NLTC)
Other Tivoli Laboratories

Trademarks

The following names have been adopted by the International Business Machines Corporation as trademarks in the United States and/or other countries and maybe used throughout this document:

IBM
Tivoli
S/390
AIX
DB2

Windows, Windows NT, Windows 2000 are registered trademarks of Microsoft Corporation.

Solaris is a registered trademark of Sun Microsystems.

HP-UX is a registered trademark of Hewlett-Packard Company.

Acronyms and Terminology

Acronyms are used to streamline text in this document. Acronyms are not always defined after their first usage, however. Reasons for this include: many of the acronyms are commonly used and known by the intended audience of this document, the purpose of this document is primarily reference, and a glossary is provided to define new and less commonly used acronyms and terms.

Types of Notes

The following types of notes are used in this document:

Note:

Either explains aspects of a definition that may not be obvious to the reader or provides auxiliary information about it.

Design Note:

Design questions/notes/issues are contained in boxes as is this text.

Table Of Contents

Section 1. SANergy System Overview

1.1 Background

SANergy software enables multiple host computers to directly attach to storage devices and safely share file data. Without SANergy (or similar software), the machines would each believe that all the storage was theirs and only theirs, and thus not coordinate each other's caches, overwrite allocations, mark or otherwise write to disks when unauthorized, and rollback (undo) transactions being performed by another machine. SANergy solves those problems plus allows heterogeneous systems to share data. The fundamental approach of SANergy is to capitalize on traditional LAN networking for the majority of the management of "sharing" activity (metadata), but use the SAN fabric for the actual file data I/O.

SANergy does not introduce a new file system to the world. Rather, it takes existing file systems and adds some layers of software to provide the correct behavior in a shared-SAN file system environment. Each file system in the SAN has a single machine that is the true file system owner, referred to as a Meta Data Controller (MDC). The other participants are referred to as SANergy clients (of *that* file system). To date, SANergy only supports a small number of file systems and MDC platforms. The purpose of this document is to fully describe the functionality that is need by a particular file system in order for SANergy to consider using it as a SANergy-supported one.

Note:

At various times and in various documents, the terms "volume" and "file system" have been used interchangeably. Either term is correct and there is no implied distinction. Other terms, such as "disk", are very specific and always mean a "physical amount of storage that appears to the host computer as a disk" (and **never** means or implies a file system or volume).

1.2 MDC's and Clients

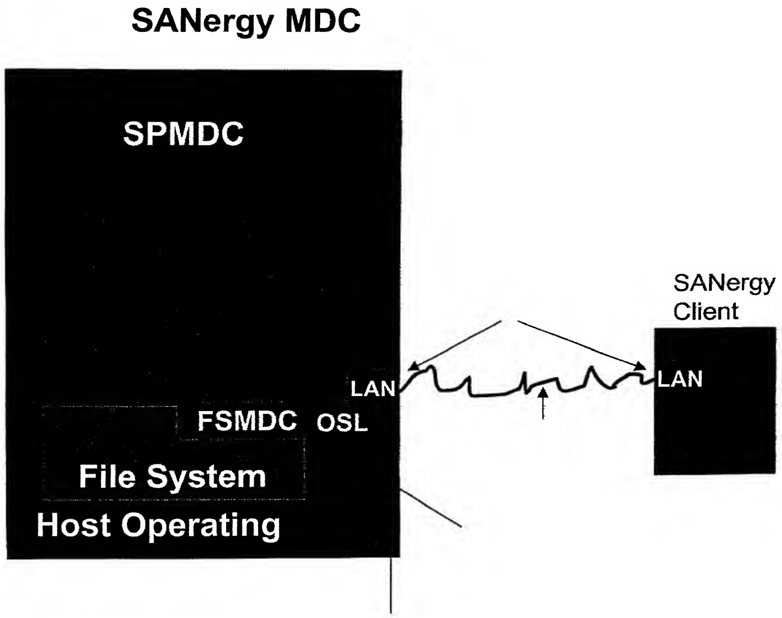
A SANergy system consists of two distinct types of machines playing in the game: MDC's (metadata controllers) and Client systems. The terms MDC and Client are a per-logical-file-system basis. That is, a physical machine could be an MDC for one or two volumes, and a Client to the rest, or any other combination (there is only a single MDC per volume but no limit to the number of clients).

The MDC machine is the only machine that hard-mounts (read-write) the volume and lays down a file system and manages that file system with respect to authentication, security, administration, and allocation. The philosophy of SANergy is to exploit existing operating system platforms and their file systems by making them work correctly in a SAN data sharing world. In one respect, an MDC is somewhat like a file-server for a volume; other machines use conventional networking to access the volume and thus are authenticated and controlled in the traditional networking model. However, only meta-data (file open, security, allocation information, etc.) take place in the SANergy approach; the actual file payload (data) is transferred directly between the storage elements and the accessing computers.

The Client machines use ordinary networking to mount the volumes and open files. The SANergy code on the Client intercepts reads and writes and issues those directly to the disk elements. It is able to do this because it has coordinated with the MDC (regarding caching) and asked the MDC for the list of physical blocks for the file (aka "the map").

Determining the block list and coordinating the access to the blocks is very file system and platform specific. SANergy supports a few different MDC platforms directly, and the goal of this document is to allow other file systems and platforms to build interfaces that work correctly in a SANergy environment.

1.3 Architecture Overwiew



File system providers need to provide the functionality of the box labeled "FSMDC". This document details that exact API including the rules and responsibilities as well as the programming interfaces. The SANergy Protocol (SP) describes the LAN interface between an MDC and a client (data structures and functionality). Depending on the host platform and various business arrangements, the main box (SPMDC) could be built by the SANergy team or by the File system and/or operating system vendor.

There are separate documents for all the other functional layers. This document details the FSMDC layer only.

1.4 References - Applicable Documents

There are a number of applicable documents for creating full MDC.

- OSL Document: Describes the Operating Specific Layers and services needed by higher levels of the MDC.
- SP Document: Describes the SANergy Protocol that goes across the LAN wire. All the functions, rules, and obligations are also described.
- LAN Document: Describes base LAN layer functionality.

Section 2. FSMDC Overview

2.0 Introduction

There are two basic capabilities that the FSMDC must provide:

- Provide "volume-to-disk" translations.
- Provide file disk map information.

Ultimately, SANergy clients access a file's data by reading and writing directly to physical storage. Thus, a SANergy client must be told the exact physical information regarding the file. This is done in a two-tier hierarchy: a physical description of the physical disks that comprise a logical volume and then the description of the file blocks relative to that volume information.

There are many details regarding how long such transmitted description information is valid and what steps are necessary to protect the overall "system" integrity of the data (i.e. caches).

SANergy makes an attempt to simplify the overall task of allowing multiple computers to share physical storage directly. In fact, the patented approach is novel enough that most people admire the overall simplicity. However, it is often over simplified in the SANergy marketing and presentation literature. There are quite a few rules and responsibilities of a properly coded MDC and those regulations must be strictly watched. The most important rule is regarding proper flushing and purging of MDC caches. File systems in general have always had an ability to "flush" dirty data to storage, but there is a new and more important need with a SAN system: purging. Purging means removing pages from the MDC memory system because those corresponding pages on disk might have been written by another machine.

A properly written FSMDC will provide for 100% safe, reliable, coherent, and recoverable byte-level data sharing between multiply attached hosts.

2.1 Packaging and Co-Existence

"fsmdc.h" describes the functions, parameters, and codes necessary for this interface layer. These functions must be provided by the file system manufacturer as the functions are very file system specific. They need to be wrapped-up into a single shared-object library which can be dynamically loaded by the higher level SANergy MDC code at runtime. It is entirely possible that on a single MDC there may be more than one file system present and thus more than one of these libraries. The layers above the FSMDC are designed to exhaustively load all detected FSMDC modules (shared libraries), this way new file system support can occur independent of SANergy releases.

2.2 Names, Cookies, and Locks

2.2.1 Names

All functions throughout the SANergy layers that accept names for files, volumes, shares, or anything else, use the NT-style UTF-16 UNICODE encoding. It is imperative that this requirement be recognized by the various file system FSMDC providers as SANergy lives in a multinational world. If a given file system does not support UNICODE names internally, it must do the best job possible to translate the incoming names appropriately.

2.2.2 Cookies

The FSMDC layer introduces a concept of a "file system specific and unique identifier" called *cookies* used for referencing both Files and Volumes. It is possible for a file while in use to have its name changed or even deleted. It is also possible for multiple separate files to share a common name. The use of cookies avoids any confusion.

Before higher layers of SANergy can manipulate a file (extend the map, set a file size, etc.), it must first ask for a Cookie for that file (by name). All subsequent operations on a given "session" of a file will use that cookie and never the name again. Any and all reports from the FSMDC layer regarding a file (such as the allocation map) will also report back the associated volume cookie. Volumes can also change in definition while the file system is up and running (e.g. a volume can be extended, striping changed, or other physical disk members added). Any change to a volume MUST result in a different cookie for the volume. Thus, dynamically, SANergy will be able to detect any material changes to the file system and files (even while files are in-use).

2.2.3 Locks and Leases

One of the most important functions provided by the FSMDC layer is the delivery of file-map information (the physical description of all the blocks that comprise a given file). This information has a life-span associated with it ("lease"). The SPMDC layer decides how long a map is considered useable and reports back that information along with the map to the caller. The caller (SANergy client) is obligated to ***never*** reference any part of the map or the blocks it represents beyond the lease duration.

While a lease is active the file's allocated blocks are considered "locked". The FSMDC guarantees that there will be no removal or reclamation of any of those blocks of the file (note, the current file-size is a separate concept and somewhat unrelated to whether blocks associated with a file are still associated with the file). An "unlock()" function is provided to allow the higher layers to inform the FSMDC layer that the file map is no longer being referenced and thus the FSMDC layer is now free to reclaim any blocks or completely reorganize a file (perhaps defragging). If the lease on the file expires it is *exactly* as if the a call to "unlock()" was made.

The FSMDC does not need to keep "counters" regarding how many times a particular file was locked. A single "unlock()" will unlock the file. The higher levels of software (SPMDC) will track how many parties are active on a file and issue or defer issuing the unlock() call.

The SANergy client will provide a desired lease value to the SPMDC layer. It is typical for SANergy to ask for very short leases (i.e. 5 seconds) for files opened for read-only access. Files opened for read-write access SANergy is likely to request a much longer lease. To save overhead, SANergy clients will typically let leases expire rather than specifically "close" the session (particularly for read-only files).

To review, the FSMDC layer concerns itself with delivering file maps and considers the allocated blocks of a file "locked" until the unlock() function is called. The SPMDC layer has a concept of leases and manages calling the unlock() function as needed.

2.2.4 File Size and Extended Size

With SANergy, a file has two size values: the current reported file length (File Size) and the Allocation Size. For efficiency, when creating or extending a file, SANergy will ask that the file grow by a very large amount (100MB for instance). It is assumed that the cost of allocating a small amount of space is about the same (overall) as allocating a larger amount, and minimizing the number of times a file is extended is extremely important (LAN transmissions and related issues).

A file's size may be changed at any time to any value by the local MDC or any SANergy clients. The Allocation size, however, is only allowed to *grow*, until there are no locked-maps active. That is, if the MDC provides a locked-map to a requester, it is obligated to never reclaim and reassign those blocks until the file is unlocked (regardless if the request comes from a local process on the MDC or a SANergy client).

The MDC should report "file size" to any inquiring application, and *not* the allocation size. The SANergy GetMap() function allows the MDC to report both values. Similarly, there are functions provided by the FSMDC layer to allow SANergy change either or both values.

2.2.5 Volumes

SANergy uses a two level hierarchy for fully describing where all the disks blocks are that comprise a given file. A file-map in the SANergy sense is a map of blocks relative to a prior disclosed volume. A volume is an aggregate of physical disks, offsets and pieces of those disks, and possibly an algorithm (e.g. NT RAID-0 striping) over the disks. A volume is referred to as a "glom" in fsmdc documentation which implies that there is some algorithm applied to the set of disks when interpreting the file-map information (typically some striping algorithm or spanning).

It is not required for any SANergy client system to run the same volume manager as the MDC. To be exact, the clients go below any volume manager anyway to access the disks. This is to allow the widest range of computers to connect and successfully run in a SANergy environment.

Using the two layer abstraction allows for far more compact map descriptions. Without a volume abstraction, describing a large file that is striped across several drives could result in a map that is many times larger than in this approach.

SANergy is designed to work correctly in any disk environment and does not count on world-wide-names of devices (that Fibre Channel allows). Thus, for a client system to locate a particular disk that the MDC references, it is sometimes necessary to perform a search. The FSMDC must describe to the clients a unique identifier somewhere on each disk (such as a disk signature). The API provides for the MDC informing the clients exactly what the offset of the identifier is and how many bytes long it is. Clients will quickly scan the disks and typically build a table of this information.

3.0 API's

This chapter will provide details of each and every API the FSMDC layer must provide. It will sometimes show particular data structures and function parameters, however, this document might be rev'd differently from the actual "fsmdc.h" header file. The header file is the definitive definition and this chapter shall serve as a guide and offer supporting verbiage (but might be slightly inaccurate over time).

Each FSMDC implementation is required to provide all of the functions described in this section and the header file. The layers above the FSMDC will definitely be looking-up and calling each of these functions.

3.1 Data types

The "fsmdc.h" file defines many key and portable datatypes. The bulk of the FSMDC functions and definitions do not use such data types as "long" or "long long" or even "byte" or "char". All such data types have an "FS" equivalent that is designed to provide a consistent definition across all platforms (for example, a "long" on Tru-64 is 64 bits).

The actual definitions are now in a file called "fstypes.h" which is 'included' by fsmdc.h.

3.2 FS_GetVersion()

This is typically the first function to be called by a given MDC implementation. It will inform the SPMDC layer of what generation this particular FSMDC is from. Recall, the FSMDC modules can be distributed by parties that are not directly from SANergy and could thus have various ages to them.

This function returns a FSLONG value indicating the non-decimal hundred's version value. That is, version 2.20 would be reported back as "220" in decimal.

3.3 FS_GetVolCookie()

This function requests a volume "cookie" given a pathname. The pathname could be something that merely represents the disk (e.g. "c:\" or "/exports/test1") or could embody a full file name with it. The goal is to decode the volume that is referenced by the parameter and return a cookie for it (so that a subsequent call can get details about the make-up of the volume if desired).

This function might not be called frequently. It is conceivable that the caller might first call the "GetMap" function which returns a volume cookie directly.

3.4 FS_GetVolByCookie()

Given a volume cookie (either from a GetMap() or GetVolCookie() request) return the detailed information about the make-up of the volume.

The caller provides a data area for this function to write its description. It is entirely possible that the caller does not provide enough space and hence specific return value indicating this. The return value of 'msglen' will indicate the total message length (amount of data written into the buffer) or the amount of data that *would* be written into the buffer if the buffer was large enough (thus, in a condition where the caller buffer is not large enough, the caller can look at this value to determine to correct size to allocate).

This message as well as others also provide a "vendorStatus" field. Every FSMDC function is required to return an offical FS error code. Many times the FSMDC supplier has more information available to it in an error condition and this field allows the FS to disclose an FS-specific value. In the case of an error, the disks data structure can contain an UNICODE error string that details the error.

This function reports back an "array" of descriptions of the disks that comprise this particular file system, as well as particular information regarding how these disks should be treated as a volume-set. The 'GlomType" field might indicate a certain type of striping, or spanning, or something else. If a particular FSMDC has a new type of volume information, it could always merely report back the set of disks, and then later in the GetMap requests report individual (verbose-physical) descriptions for each block. In any case, this report must list all disks that could be possibly referenced by a subsequent GetMap request. The caller (a SANergy client) will seek out and locate each physical disk after making this call; if it can't find all of the (non "META") disks referenced, it will not consider this volume useable by SANergy.

- Special Note on "META" disks: The FSMDC has an ability to tell higher layers about disks that are logically part of a volume but contain no file data and those wont be ever referenced in a file-map. These are "META" disks and identified by the special attribute bit in the disk-info structure. SANergy clients should not expect to find such disks on the SAN and it is there not considered to be an error if these disks do not exist. Some file systems completely separate out file-data from file-meta-data and store all the meta data on physically separate disks (for performance), yet these "META" disks are still considered part of the overall volume. Those meta disks could be local/internal disks on the MDC, and thus not visible on the SAN.

The 'BlockSize' field is used by the SANergy clients to determine the smallest possible I/O that can correctly be issued to the storage device. Typically this value is 512.

Disk Details

The FSDISK structure details the information necessary for SANergy clients to locate and use the disk correctly. Up to 16 bytes of "unique" data is available for the FSMDC to specify a given disk (DiskID). This ID can live at any arbitrary byte offset on the physical device. An additional value (???) indicates just how long this particular value is for this FSMDC.

The BlockOffset field states where the (block wise) on the device this particular partition/slice starts. The subsequent file maps will be relative to this value, and not to the physical block-0 of the device. The nBlocks field indicates how blocks beyond the "BlockOffset" one could possibly be referenced. SANergy will consider it an error if a map ever references a block beyond this range.

3.5 FS_GetCookies()

Before a file-map can be requested for a file, a set of cookies for that file must be learned. Recall, cookies are unique and are persistent across deletes or renamings of files or volumes (that is, they are never reused). This function accepts a full pathname and reports back the associated volume and file cookies.

If the caller is not familiar with the volume cookie (possibly because it is the first time this volume is referenced, or possibly this is the first time since this volume has been extended or otherwise altered), it will then call the GetVolByCookie() function learn about the details.

3.6 FS_GetLockedMap()

The main work-horse function of an MDC is the GetMap function (actually called "FS_GetLockedMap()" but often referred to as simply GetMap()).

Maps can be potentially quite large, plus, a file might not be fully resident (in the case of an HSM). The caller requests for a range of information regarding a file, and this function may return that range or a sub-set of that range (providing for HSM issues).

As with the GetVol function, the caller specifies an output area to hold the variable-length map description. It is possible that the caller has not provided enough space and thus there is a specific error message indicating this.

The caller can specify a set of behavioral flags. The "WAIT" flag tells the FSMDC that the caller wants the entire requested range and will not tolerate a partial map. This is sometimes needed in the case of an HSM situation; the caller might have already used the initially reported map and now needs the whole map regardless of how long it takes. The "NO_HOLE" flag says that the caller expects that the return map contain all useable blocks and that "holes" are not permitted. In such a case, the FSMDC must first create real storage for such areas.

There is flexibility in describing a file map. The FSMDC must choose the most compact method that correctly represents the file (the difference could be several thousand bytes different and where this information is typically transmitted over LAN, it is critical to use the most compact method available).

3.6.1 Lock

When this function is called, the FSMDC is required to "lock" the allocated range of a file, which means that under no circumstances should the file's allocation size be permitted to shrink (even if a local application on the FSMDC truncates or removes the file). The blocks must remain assigned to this file until the subsequent FS_UnLock() function is called for this file.

3.6.2 Partial and sub-maps

It is possible for this function to return only a sub-range of the map-range requested (return code FS_M_ SUBMAP). This is to allow for file systems were it can take a great deal of time to build the entire extent list and it is assumed that the caller can start consuming the file with a partial map. This is also useful with HSM products where some small portion of a file might be resident, but the rest needs to be fetched from backing store. The caller can re-call this API to ask for the remaining range and possibly specify the FS_WAIT flag. This is not to be confused with the return code of FS_M_PARTIAL which means that the caller did not allocate enough space to hold the entire information.

3.6.2.1 Glom Or Physical

File maps are returned as a list of "extents". There are a number of different extent-types defined in the fsmdc.h file. As mentioned above, the MDC is required to return the map in the most compact type possible (for instance, an ordinary file that has no holes and no special regions and is described in its entirety can use the very compact FSEXTENT_COMPACT_GLOM type. The difference in size of the various extent data structure choices is quite large (more than double) and thus the total map transmitted could be quite different size-wise.

Most of the extent types are glom-relative (and with the _GLOM identifier). This means that the client side uses specific knowledge (i.e. about striping and striping algorithms) to infer the physical disks that underlay this extent. One particular extent type, FSEXTENT_VERBOSE_PHYSICAL is **not** glom-relative and instead each extent references a specific disk. This is a catch-all to allow clients that do not understand certain MDC algorithms to still be able to get useable map information (at the expense of the map being larger). A client can specifically ask for this extent type by specifying the FS_M_FLAG_VERBOSE_PHYSICAL flag on the request.

3.6.2.2 Disk Ordinals

Some of the extent types have a "disk ordinal" value associated with each extent. This field has various meanings based on the overall map-type returned. For "compact", there is no such field present as the glom type is sufficiently straightforward that the client will know which disk member(s) underlay this extent. For the _SIMPLE_GLOM and _VERBOSE_GLOM types this value will indicate a glom-specific "start" value (if needed, and some map-types such as FS_GLOM_SIMPLE wont need it). For the _VERBOSE_PHYSICAL extent types, this value is the disk that this particular extent underlays and there is no implicit tie-in to other disks).

3.6.3 Validity of a block

The 'validity' field of the verbose map indicates whether a client can directly access a particular extent. The "DIRECT_ACCESS" flag indicates that it is safe for a client to go directly to storage for this data (which is the typical case). Other values should not be directly accessed. The "HOLE" flag indicates that this extent represent a logical "hole" in the file. In such a case, the client can simulate zeros for "read" operations. For "write" operations, the client must either re-ask for the map and specify the "NO_HOLE" flag or send the "write" over the conventional technology (LAN). For "COMPRESSED" or "ENCRYPTED" the client will also send the request over the conventional technology (unless it knows for sure what the encryption or compression technology is).

3.6.4 Purging

Whenever this function is called, the FSMDC layer is **required** to both flush and purge any data related to this file from its buffer caches. It is CRITICAL that the FSMDC purge the blocks such that any subsequent local references to the file data result in the data being retrieved from disk. If this requirement is not followed, then the file will become incoherent and applications will receive the wrong contents.

3.6.5 Byte Range Locks

The FSMDC is responsible for honoring file locks (byte range lock) requests from local and network applications. SANergy clients will issue ordinary network byte-range-lock requests.

It is **required** that the FSMDC flush and purge the corresponding lock region any time a byte-range-lock request is made and the file is currently locked (meaning that a map has been served). If it does not do this, corruption will result.

3.6.6 Forced Verbose Maps

The caller can specify the FS_M_FLAG_VERBOSE_PHYSICAL bit in the flags parameter which means that the caller insists on receiving back a map in the verbose-physical format. It is critical that this flag be supported. The reason is that not all SANergy clients will have an inherent ability to understand all glom types. New glom types will come along, and/or older SANergy clients will be in the field and might not understand certain glom types. When a SANergy client receives a map that has a glom type it does not support, it will re-ask for the map with this bit set indicating that it will accept the lengthy verbose *physical* description instead (which every client is required to handle).

3.7 FS_UnlockMap()

This simple function merely unlocks the specified file. After this function completes, the FSMDC is free to reallocate, reassign, or reclaim any of the blocks of this file. There will be only a single UnlockMap() issued for a file regardless of how many "GetLockedMap()" calls have been made.

3.8 FS_SetFileSizes()

Any file that is open for writing may indeed need to change either the current file size or the allocated size or both. For efficiency, SANergy clients will over-allocate space for a file so that it does not have to ask for space for each and every write operation. This is called "hyper allocation". When the file is unlocked (via FS_UnlockMap()), the FSMDC is free to reclaim the over-allocated space (until a call to Unlock is made, the FSMDC promises to not reorganize or deallocate the blocks that comprise the file).

This function is used to both inform the FSMDC of a file's current file size as it is growing, as well as to increase the over-allocation space. It will never cause a reduction in the allocated space as that can only happen by the FSMDC itself, and only once the file is in an unlocked state.

When this function is called specifying a larger allocation size than is current (an "extend" operation), the FSMDC must allocate the amount specified and ensure that there is "real storage" behind the allocations (thus "not sparce"). If it cannot perform the allocation (as in a disk-full situation) it must attempt to acquire as much as possible. The response will indicate if the new allocation size succeeded and the client-side will interpret any shortage as a disk-full situation. SANergy Clients routinely "hyper extend"; that is, over allocate a file by a great many megabytes while the file is growing.

When the MDC is requested to extend a file, it should not zero-out the newly assigned blocks. The client side owns this responsibility (and this leads to much higher performance).

Any time this API is successfully executed, the file's "modification date" needs to be set to the current time. This allows clients that are streaming/growing a file to have an ability to effect the date, which other clients may wish to observe (particularly important for backup and HSM applications).

Specifying file size as negative one means to **not** change or set the file size at all (but still have the by-product behavior of considering the file changed as of the moment of this call, for reporting and inquiry purposes).

Specifying AllocSized to –1 indicates that no change is desired to the allocation size (presumably there is a non-negative-1 FileSize specified).

Attempting to decrease the allocation quantity is benign; allocations will only be discarded once a file map is unlocked. File-size can be set to any value between negative-1 and the current allocation size (greater than the allocation size is an error).

The 'flags' value is for future use, possibly to describe to the FS what type of blocks to allocate or how contiguous or similar.

The 'minalloc' size tells the FSMDC layer the absolute minimum size needed and if that can't be allocated then return an error. In general, the caller wants the 'allocsize' allocated, but in a near disk full situation might be willing to settle for less.

Files that are multiply opened for interleaved append operations pose special difficulty. It is possible for each client to have a different opinion of the current file size. Although the resultant interleaving of data might be somewhat different than in a non-SANergy case, the file should still be "correct" (that is, there should be no uninitialized data and no extraneous data). Even with SANergy, network caching and flushing can result in odd interleaving. Thus, if a 100% correctly interleaved data is desired, the applications should be using byte-range locks to correctly coordinate their activity.

APPENDIX B

Patent Application for

LOW OVERHEAD METHODS AND APPARATUS FOR SHARED ACCESS STORAGE DEVICES

SANergy SP Specification

Version 2.2.3

22 June 2000

Author & Technical Contact:
Chris Stakutis, CTO SANergy
SANergy, Westford MA
Tivoli Storage Business Unit
617 693 9770
cstakutis@tivoli.com

NOTE:

The hardcopy version of this document is FOR REFERENCE ONLY.

It is the responsibility of the user to ensure that this is the current version. Any outdated hardcopy is invalid and must be removed from possible use. It is also the responsibility of the user to ensure the completeness of this document prior to use.

About This Document

This document is the Specifications for the **FSMDC** layer of the SANergy SAN-based file system sharing product.

Who Uses This Document

3rd party customers and partners
Strategy & Architecture
Technical Office(s)
Designers
Developers
Function Test
System Test
Performance
Legal Counsel
National Language Technical Center (NLTC)
Other Tivoli Laboratories

Trademarks

The following names have been adopted by the International Business Machines Corporation as trademarks in the United States and/or other countries and maybe used throughout this document:

IBM
Tivoli
S/390
AIX
DB2

Windows, Windows NT, Windows 2000 are registered trademarks of Microsoft Corporation.

Solaris is a registered trademark of Sun Microsystems.

HP-UX is a registered trademark of Hewlett-Packard Company.

Acronyms and Terminology

Acronyms are used to streamline text in this document. Acronyms are not always defined after their first usage, however. Reasons for this include: many of the acronyms are commonly used and known by the intended audience of this document, the purpose of this document is primarily reference, and a glossary is provided to define new and less commonly used acronyms and terms.

Types of Notes

The following types of notes are used in this document:

Note:

Either explains aspects of a definition that may not be obvious to the reader or provides auxiliary information about it.

Design Note:

Design questions/notes/issues are contained in boxes as is this text.

Table Of Contents

Section 1. SANergy SP Overview

1.1 Background

The **SANergy** protocol (SP) is the language used by any two SANergy-enabled computers to communicate and coordinate activities. For review, recall that the FSMDC API is the layer of software on an MDC that sits closest to the proprietary file system. The FSMDC layer provides a common interface for generalized SANergy-related MDC file system services. The layer above that provides the true SANergy-related MDC "services" (the activities an MDC must do in order to be a true MDC, which it will use the functions in the lower FSMDC layer to achieve these activities). The MDC provides those services ultimately to SANergy client systems. This document describes the RPC details as well as additional rules and responsibilities of those services.

Note: The reader must first read and fully understand the FSMDC spec before attempting to understand this layer. The implementer of an MDC must adhere to all the rules and responsibilities outlined in the FSMDC document.

The messages between the SANergy client and MDC system are currently one direction only; from client to MDC (with a response, always). The MDC never sends an unsolicited message to a client. Each and every LAN transmission has an encoded "revision" number in the message. This allows the two sides to be a different revision levels and to deal with that accordingly. It is imperative that both sides check and respect the version numbers.

This document, as with the FSMDC one, avoids showing exact data structure or API definitions. The accompanying header file (sp.h) is the definitive source for the details. The purpose of this document is to provide sufficient verbiage and explanations to correctly implement the SP protocol.

1.2 Transports

There are currently two methods of transporting the messages between the client and MDC: the out-of-band IP datagram, and the ioctl() technique. The SP protocol is rich enough to support both of these transports, although the information inside the messages might vary slightly based on the transport in use.

Datagram

Datagram messages are sent to port XXXX (defined in the sp.h header file) of the MDC, and the ephemeral port is used to carry the reply. Datagrams are chosen for their speed and simplicity. The MDC-side should be designed for processing many requests in parallel from many different SANergy client systems.

Ioctl()

The CIFS protocol supports an ability to send a file-system control message from client to server side (sometimes referred to as an ioctl(), or as an fscontrol()). SANergy uses ioctl number XXXX for its messages. A SANergy client that detects it is running through the CIFS protocol will first attempt the ioctl() interface. Should that fail, it will then use the Datagram interface and remember not to bother with ioctl attempts when targeting that server for future requests.

Section 2. Major Concepts

It is assumed that the reader of this document is intimately familiar with the concepts and services provided by the FSMDC layer. The SP protocol essentially bundles those functions up with additional functionality that overall embody what an MDC has to do to be an MDC.

2.1 Minimal Messages

It is a fundamental requirement to minimize the number and size of LAN transactions between SANergy client and server. Thus, some of the messages may encompass several distinct functions that are known to be needed together (and thus reducing the number of transactions). This occasionally leads to parameters that are over-loaded (multiple interpretations) or have different meanings based on the presence of other parameters.

2.2 Leases

Leases are a general SANergy MDC-to-client concept that provide a great deal of operational efficiency. Simply, a lease is a way to request access to a file map for a period of time, and when the period expires, the lease is automatically dissolved (thus, there is no need for an additional expensive message to declare termination of the need for the map). SANergy clients request a lease duration, but the MDC dictates the lease period granted.

The majority of file activities in a system are "open for read" which are then sequentially processed and closed relatively soon after being opened. With that in mind, this layer distinguishes between files opened for reading and those for writing (writing being more complicated). Very short leases are typical for read-only parties (5 or 10 seconds) whereas open-for-write parties have to be tracked typically for longer periods.

Clients are required to respect the lease values. That is, it is illegal to reference a file map (move data to or from disk based on the map) after a lease expires. When a lease is expired, the MDC is free to completely reorganize a file (provided no other leases are in process for that file). Typically, an MDC will immediately clean up any hyper-allocations of the file. At expiration time, the last known file size is considered the high-water mark of the file and the MDC is free to remove blocks between that value and the end of the allocation area. Thus, a client must be careful to apprise the MDC of any growth to a file's size ***before*** the lease on the file expires.

2.3 Tracking

The MDC needs to keep track of file maps that have been served out (and locked). It is entirely possible for a given file to be multiply "opened". The FSMDC layer does not keep counts or stacks related to how many file maps have been served. Thus, it is the responsibility of the MDC to call "UnlockMap()" only when all parties are finished with the file map (hence, it must reference count the parties to a file). This is a design and implementation issue for the MDC layer and not germane to this document, but worth mentioning at this time because the SP protocol introduces a concept of "keys" (described next) for this reason.

2.4 Keys

Each time a SANergy client starts transactions to a file (and thus receives a map for the file), the client becomes registered as a party to the file and a "key" is generated. The key is then used by the client for subsequent transactions to that file. Keys are similar to file-handles; that is, there is one-per-active-session on the file. The key values themselves are unique from a given MDC (but not unique across the SANergy system). The MDC side generates the keys and then uses them internally to track each map dealt out (for the purpose of cleaning up expired maps or looking up references from the client).

2.5 'future' fields

Nearly every LAN data structure in this document has a 'future' field which is typically 8 unsigned long words long. The purpose is to make it simpler to add capabilities or make small changes to the protocol without having to re-do the data structure. 'future' fields are guaranteed to be zero (the sender must zero-out the data structure prior to sending).

2.6 Message header and overview

Every message between the client and MDC (in either direction) has the same overall format, described by this data structure:
- *Note:* Any data structure or function definitions presented in this document are to be used for annotation and discussion purposes only and should be considered the official definitions. The official definitions are found in the appropriate header files (e.g. sp.h or lan.h).

```
typedef struct {
            FSULONG endian;
      FSULONG revision;
      char    messageID[16];
      FSLONG  functionCode;
      FSULONG length;
      FSULONG flags;
      FSLONG  result;
      FS_SYSTEM system;
} LAN_MsgHeader;
```

The above structure is the preamble for each and every message between SANergy members. The SP layer (sp.h) defines messages all of which have this data structure as the first element.

The endian field indicates the endianness of *this* message. The observer of the message will know whether or not binary data values need to be endian-adjusted by comparing the value in this field against the endian constant. If they match, no adjustment is necessary. Each and every message sent from one computer to another will be layed-down in the natural endianness of the *sending* machine. The receiving machine owns the responsibility of adjusting the data if necessary. Even "replies" to messages will be in the endianness of the machine *sending* the reply.

The 'revision' field is the critical link between the two sides so that they can be sure to speak the same dialect of this API (or reject communication if desired). As of this writing, the current version number is 220 (for 2.20) and is referred to by the constant SP_LANVERSION.

The "messageID" field is an arbitrary string that the client creates and that the MDC must use in the response (likely it is some sort of sequence information). The client ensures that each message sent to an MDC will use a unique message ID. Thus, the MDC, if it sees the same message ID twice, it can respond with the exact same previous response. Similarly, if the MDC hasn't yet responded, it will interpret the duplicate as an indication of a client's impatient ness. There is no guarantee, however, of message ID's being globally unique (that is, multiple clients might use the same algorithm/counter to generate the Ids). Clients can't tell ahead of time how long it will take an MDC to process a get-map (or any) request (the time could vary from a millisecond up to several minutes in the case of an HSM fetch). Thus, it is conceivable for a client to ask for a map multiple times, worrying that the MDC may have dropped or ignored the earlier requests.

The 'functionCode' describes the function/service that is being requested and dictates the layout of the data beyond this header in a packet.

The 'length' field indicates the overall amount of data transferred in this message, including this header and its associated data.

The result field is only interesting on a reply and indicates the overall status of the request. Result values live in "fserr.h".

The 'system' field is a structure that describes as much about the computer system sending the message as possible (e.g. operating system, hardware platform, file system syntax for path separators, and other information). This structure needs to be filled-in on both the sending and replying operations so that the other side can consider behaving differently based on who it is talking with.

Responses

The response back from the MDC to client will be the same overall format (revision, ID, function, length, and then an arbitrary size body). The function code for a response is **always** the negative of the incoming message function code. There are unique responses structures defined for each and every function and are located in the header file below each outbound structure.

2.7 Names

As with the FSMDC layer, any and ALL "name" (or string) type data fields are encoded in the NT style UTF-16 format (never ASCIZ). All such strings will be terminated with a 16-bit zero value (thus the length of the string can be learned by scanning).

Section 3. The Functions

This section will discuss each and every API that can be issued from the client to the MDC (there are currently only five functions). The reader should match up the descriptions in this document against the sp.h header file (this document avoids showing specific definitions as those could slightly change and not effect the overall descriptions in this document.

3.1 LANGetVolume

SANergy clients will need to know the specifics of a volume (file system) regarding physical disks and offsets and sizes and so forth. That information ultimate comes from the MDC's FSMDC layer and there is an SP interface for it, with some differences in the parameters.

The SP_LANGetVolume structure is used to request this information. The 'functionCode' field will be set to the constant SP_LANGETVOLUME. The response structure is SP_LANGetVolumeResp.

A SANergy client possibly needs to be authenticated (license-wise) by an MDC before the MDC should provide services to it. It is convenient to do that authentication at the time of requesting volume information (because volume information must be requested before any useful other SANergy activity can take place). The 'license' field contains the client-side SANergy software license string. The MDC is required to validate that this license is valid, not duplicated (if that is to be prohibited), and so forth. The MDC can reject providing volume information with the FS_E_LICENSE error code. If an MDC does not know how to validate or track licenses, it can return "success".

If the pathname provided is zero-length (either no data or contains a zero-length Unicode string), then the 'cookie' value is considered significant (and this will call GetVolByCookie()). The 'pathname' for this function is the export name or the share, but **not** a true file name or mount point. The implementer of this function will need to translate that share/export name into a real mount point so that the GetVolCookie function can be called correctly. If the translation could not succeed, this function should return one of the SHARE_NAME error values. If this command is activated through the IOCTL interface, then the pathname and cookie fields can be omitted as it is implied by the file object that it is riding on (typically, then, *some* file will have to first be opened, possibly a directory or dummy file on the volume, or similar).

Category Error return values:

- Share name translation (FS_E_SHARExxx)
- License errors (FS_E_LICENSExxx)
- FSMDC errors (from the call to GetVolCookie or GetVolByCookie)
- Misc errors

3.2 LANGetInitialMap

The SANergy client will need an ability to get a physical map for a given file. There are different modes or reasons why a SANergy client will need that information and there are a couple of LAN interfaces created optimized for the various needs.

A new concept is introduced: "intention". The purpose of "intention" is for the client to help the MDC optimize various management tasks by informing the MDC of whether the client will do strictly and only "reads" of a file versus other random activities (writing and stretching a file).

A SANergy client will need maps at various times in a file's "open" life span. The information that the client side has available and its needs are different accordingly. The LANGetInitialMap function is tuned to the needs of a file just being opened by a client for the first time this session (versus a lease expiration or an extend).

The client should call LANClose when it knows that it no longer needs access to map and thus the MDC can start any cleanup operation. Typically this is only needed when a lease period granted is substantially long. For short leases, the kind that are typically granted for a READONLY open for instance, it is more efficient to let the lease just expire (saving an additional LAN transmission).

The data structure SP_LANGetInitialMap defines the input to this function and the response structure is SP_LANGetInitialMapResp. Refer to those definitions in the header file while reading the rest of this section.

The pathname is a two-part UNICODE string (the first part being the share or export name, and the second piece being the relative filename). The first part is null terminated (in UNICODE) and the second part follows immediately (this facilitates having single variable length field). The pathname can be omitted in the case of using the IOCTL interface as the file-object used for the communication implies the specific file.

'filesize' and 'allocsize' can be set to desired initial values or –1 meaning that no change to the existing value is desired. Files opened for some sort of writing would typically like to insist that there are blocks immediately allocated for the file (to save having to request blocks later). Similarly, files opened for "truncate" (or "create new") would like to initially force the file's size to be set to 0 instead of having to issue a separate set-file-size request.

To save LAN transmission, the overall pathname field is considered variable length (thus the last parameter in the structure).

The response typically is a file map in addition to a lease-granted value, the various cookies, and a "key" value. The implementer of this function generates a unique key value and the caller will use that key value for all subsequent communications regarding this *session* of the file. The key is considered "valid" only for the period of the lease. It is not legal to use a key value beyond the lease expiration.

Error return values:

- Results from FS_SetSizes or FS_GetLockedMap()
- Possible out-of-memory conditions.
- File system not supported

3.3 LANGetExtendedMap

While a client is writing new data to a file, it may run out of space from the initial allocation map. The LANGetExtendedMap function requests additional space for the file. The SP_LANGetExtendedMap structure (and ...Resp) is used for this function. The implementation is primarily built on top of the FS_SetSizes() and FS_GetLockedMap() API's. It is unlikely to ever need to stretch a file and not immediately need the new map, which is why this single function will do both operations.

The majority of the parameters for this function are identical to the LANGetInitialMap one. The 'key' is the key value returned by the LANGetInitialMap function (if this file's previous lease has expired, the caller must start over again by calling the LANGetInitialMap function). The remaining parameters are used for calling FS_SetSizes() and FS_GetLockedMap() and have the same rules and responsibilities.

The response is a new map, a new lease value, and a result code.

Error return values:

- Results from FS_SetSizes or FS_GetLockedMap()
- Possible out-of-memory conditions.
- FS_E_KEY_INV : The specified key is unknown, invalid, or possibly expired.

3.4 LANSetSize

It may be desired for a client to merely inform the MDC of a file size change (due to growth while streaming or some other adjustment via a set-end-of-file type of request). As long as the new file size is within the currently "leased" map's allocation area, there is no need for the client to receive new map information (thus more efficient on the MDC and LAN bandwidth). The LANSetSize API (with the SP_LANSetSize data structure) is for this purpose.

The parameters are straightforward. The 'key' value is the value returned from a call to LANGetInitialMap. The 'filesize' value is the new desired file size. The response is even simpler: nothing but a return code value which will hold the value returned by calling FS_SetSizes().

Error return values:

- FS_E_KEY_INV : The specified key is unknown, invalid, or possibly expired.
- Or any value from FS_SetSizes().
- Possibly a "misc" error if the size specified is considered illegal (maybe it is beyond even the allocated size).

3.5 LANClose

A client can express its intention to no longer reference a file (and thus its map). The value of doing this is to allow the MDC to be more efficient and clean up resources associated with tracking leases. For short leases (5 seconds or so) it is typically better to let the lease expire without calling this function (the LAN overhead out weighs the benefit). Longer leases, like those granted for "writes", should call this function. Typically, when a file opened for "write" closes, the final file size needs to be told to the MDC anyway, and so this function allows both "closing" the intention on a file at the same time setting the final file size.

The 'key' value indicates which file is to be closed. Typically this function is called when a file opened for write is closing, and thus it is convenient to be able to specify a final file size (instead of also having to call the LANSetSize function). The 'filesize' parameter can be set to –1 indicating that no change of file size is desired.

Error return values:

- FS_E_KEY_INV : The specified key is unknown, invalid, or possibly expired.
- Or any value from FS_SetSizes().
- Possibly a "misc" error if the size specified is considered illegal (maybe it is beyond even the allocated size).

The client must not use the file key or the file map after calling this function as the MDC is free to tear down all data structures regarding that key.